# Taming Active Content

Abstract: The private and public sectors depend heavily upon information technology (IT) systems to perform essential, mission-critical functions. As technology improves to provide new capabilities and features, new vulnerabilities are often introduced as well. Organizations implementing and using advanced technologies, therefore, must be increasingly on guard. One such emerging technology is *active content*. Although the term has different connotations among individuals, it is used here in its broadest sense to refer to electronic documents that, unlike ASCII character documents of the past, are able to automatically carry out or trigger actions without the intervention of a user. Examples of active content include Postscript documents, Java applets, JavaScript, word processing macros, spreadsheet formulas, and executable electronic mail attachments. Taken to its extreme, active content becomes, in effect, a delivery mechanism for mobile code. The purpose of this paper is to provide an overview of this topic and its underlying technologies so that the reader is made aware of the associated security risks and can make an informed IT security decision on its application. The paper provides real-world examples involving commonly available products and development tools as a way of increasing the understanding and awareness of the potential risks involved.

## Introduction

Having the ability to download files and electronic documents off the Internet is a useful function and a common practice for many people today. Web pages serve as an electronic counterpart to paper documents such as forms, brochures, magazines, and newspapers. Although paper documents come in different shapes and sizes, they are composed entirely of text and graphics. Similarly, most Web pages consist mainly of text and graphics. However, unlike paper documents Web pages can involve active content, capable of delivering electronic documents containing multimedia formats.

Active content involves a host of new technology such as built-in macro processing, scripting languages, and virtual machines, which blur the distinctions between program and data. Electronic documents have evolved to the point that they are themselves programs, or contain programs that can be self-triggered. Loading a document into a word processor can produce the same effect as executing a program, requiring appropriate caution to be taken. The trend towards active content has been spurred by the popularity of the World Wide Web (WWW). A dynamic weather map, a stock ticker, and live camera views or programmed broadcasts appearing on a Web page are common examples of use of this technology. Like any technology, active content can provide a useful capability, but can also become a source of vulnerability for an attacker to exploit.

Despite these capabilities, people tend to use electronic documents in much the same way that they use paper documents. That is, Web pages delivered from Web servers to individuals via Web browsers impart an inherent document metaphor [Ven99]. One advantage is that people are familiar with handling documents in business transactions and leisure activities, and can quickly adapt to using electronic facsimiles. Literate individuals universally understand the three basic operations with Web pages – fetching content, following links to continuations or other content, and filling in forms. One drawback, however, is that the document metaphor is generally considered non-threatening and can lull one into a false sense of security.

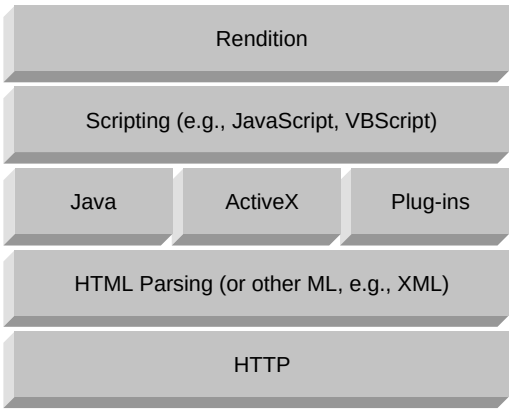
## Browser Architecture

A *browser* is the generic term used to refer to software that lets individuals view pages from various sources, including Web servers on the Internet, which make up the WWW. Netscape Navigator and Microsoft Internet Explorer are two popular Web browsers that aid in navigating text, graphics, hyperlinks, audio, video, and other multimedia information and services on the WWW. Browsers rely mainly on a simple, request-response

communications protocol, the HyperText Transfer Protocol (HTTP) [HTTP], to access Web servers. The browser requests information from a specific Web site, by sending a method request to the Web server conveying the Universal Resource Locator (URL) of the desired resource (e.g., a Web page), client information, and content handling capabilities. The URL is used to locate the server and services as a unique address of the resource.

Upon issuing the request, the browser expects a response from the server, containing a status code, meta-information about the resource, the content corresponding to the resource requested (e.g., the Web page specified by the URL), and an indication of content encoding. Five general classes of response exist, as indicated by the first digit of the status code: 1xx-informational, 2xx-success, 3xx-redirection, 4xx-client error, 5xx-server error. For example, most of us have received a 400 series code, the 404 code, at one time or another when unsuccessfully attempting to reach a site.

Both browsers and Web servers are aware of Multipurpose Internet Mail Extensions (MIME) [MIME] content types and use them during content negotiation. MIME was designed originally as an extensible mechanism for electronic mail, using the convention of content type/subtype pairs to fully specify the native representation or encoding of associated data. Content negotiation is part of the request-response protocol; the browser states its preferences and capabilities in its request and the server responds with resource representations that best match them. Content types include Audio - for transmitting audio or voice data; Application - used to transmit application data or binary data; Image - for transmitting still image (picture) data; Message - for encapsulating another mail message; Multipart - used to combine several body parts, possibly of differing types of data, into a single message; Text - used to represent textual information in a number of character sets and formatted text description languages in a standardized manner; and Video - for transmitting video or moving image data, possibly with audio as part of the composite video data format.




• Figure 1: Basic Components of a Generic Browser

The original HyperText Markup Language (HTML) [HTML4] specification signaled the birth of the WWW, and as such, it inherently stipulates the basic requirements of a browser. A browser interprets HTML control tags within Web pages, which indicate the structure of the data (e.g., beginning of item, end of item) and the way to render it (e.g., heading, subheading, paragraph, list, embedded image). The codes may also embed URLs of additional information such as images, which entail further requests to the server to retrieve the information and complete the Web page. Like most standards, commercial implementations have tended to extend the basic requirements into proprietary areas, and sometimes ignore a basic requirement or interpret it differently than originally intended. This has led to standardization bodies such as the WWW Consortium (W3C) to evolve standards along the lines of existing implementations, and developers of Web pages to undertake measures to ensure compatibility with versions of commonly used browsers. Control codes are the subject of intense

standardization and include standards for the Cascaded Style Sheets (CSS) [CSS1, CSS2] and the eXtended Markup Language (XML) [XML1].

Browsers inherently involve many different program components both internal and external. Figure 1 illustrates the common components found in most browsers. The basic protocol machine for HTTP, a parser for HTML, and a mechanism to render simple textual and graphical content are essential core components present in all browsers. The remaining components represent mechanisms to render other forms of content. To some extent, the specific choices depend on the browser manufacturer. However, competition and market demand influence manufacturers to offer components having comparable functionality with a high degree of compatibility and uniformity. Scripting languages (e.g., JavaScript, Visual Basic (VB) Script, and JScript) are a useful means of having instructions, conveyed within the HTML from the server, executed by the browser, and require an interpreter for each language supported. Similarly, environmental components for Java, Active X, and Plug-in technology allow code external to the browser to be executed at the browser. In general, the program components of a browser, both built-in and otherwise, can be divided into the following classes.

## Program Components Incorporated Directly within the Browser

Browsers contain a significant amount of built-in functionality and typically can render a variety of content types, including text, HTML delimited text, scripting languages, Java applets and common types of image files, inherently. The associated program components are functionally internal to the browser and able to directly interpret such content. In order to keep the browser safe from sources that have varying levels of trust, the program components must take precautions against arbitrary input received. Because these programs are contained within the browser, the browser manufacturer is able impose security constraints on them. Built-in functionality is also a means for the manufacturer to distinguish its product from others in various ways such as offering proprietary extensions to standard script languages, close integration and interworking with other product offerings, and entirely new content handling capabilities.

In general, script-based languages do not incorporate an explicit security model in their design, and rely mainly on decisions taken during implementation. One noteworthy example is the implementation of secure JavaScript in Mozilla [Anu98]. The implementation uses a padded cell to control access to resources and external interfaces, prevents residual information from being retained and accessible among different contexts operating simultaneously or sequentially, and allows policy, which partitions the name space for access control purposes, to be specified independently of mechanism. Java applets are also an interesting case, because the Java Virtual Machine (JVM), the internal browser component that provides the execution environment for Java applets, involves an elaborate level of security beyond that of the browser. The default security-policy settings for JVM environment are normally established by the browser manufacturer, but can be tailored by each user.

## Program Components Installed to Extend the Browser via a Defined Interface

A significant innovation in the design of Web browsers is the ability to extend them beyond their built-in functionality. For a browser to hand off content rendering to such program components, the component must register its handling capabilities (i.e., the file extensions and MIME types it supports) with the browser when it installs, Often, these extensions require full access to the browser internals and the underlying operating system in order to accomplish their goals. Therefore, programs that extend browsers typically enjoy full function interfaces to the internals of the browsers and to the operating system. The two most common means of extending browsers, Netscape plug-ins and ActiveX controls have somewhat different security models. Microsoft ActiveX controls can require authenticated digital signatures as a prerequisite for installation, while Netscape plug-ins have no mechanism for enforcing authenticated signatures. However, once an ActiveX control is installed it has free run of the machine, whereas the plug-in is confined to the capabilities of the browser.

### Programs Launched as an Independently Executing Process by the Browser

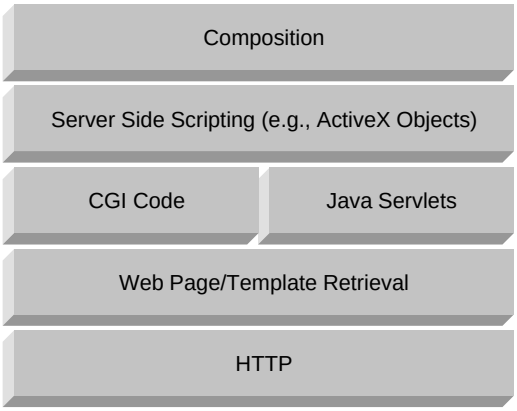
As an alternative to using a programming interface, a browser's capabilities can be extended using a so-called helper application or content viewer. Like a plug-in, the browser starts a helper application and hands off content rendering when it encounters a content type (MIME type or file extension) for which the helper application is registered to handle. Unlike a plug-in, a helper application runs separately from the browser in its own process space, and does not interact with or rely on the browser once initiated. Because a helper application runs independently, executed with the content file as input, it is complete outside of the control of the browser, including browser security controls.

### Program Components Directly Encapsulating the Browser

An interesting and somewhat unconventional approach is to embody the browser itself within another application as a means of extending functionality. The best example is that Internet Explorer, or any other browser that complies with ActiveX Container or OLE Container technology, can be run as an ActiveX Control inside an application. Visual Basic applications inherently have this capability, which allows one to not only to control the URL requested, but also interact with some of the content on the screen, or even allow HTML pages to pass information to and from the Container application [Hug99]. Microsoft MSN, America Online, and a number of free Internet service access providers configure their browsers this way. There are also a number of browsers available today that have been developed using different technology schemes, specifically for the purpose of being embedded within applications as a means of adding browsing functionality.

## Server Architecture

A Web server is a program that resides on a computer on the Internet and supplies information and services formatted in HTML or another markup language, which contains text, graphics, audio, and video content. The primary function of a Web server is to respond to requests sent to it from a Web browser via HTTP. In the simplest case, the Web server retrieves the requested resource (e.g., a Web page) from a file system and transmits it to the browser. While this approach work well for static infrequently changing information, it can be unsuitable in situations where the information is volatile, already resides in a repository under a different format, or varies according to the input provided. In such cases, the Web server responds to the request by creating the content dynamically, typically by spawning a process or lightweight thread to generate the information. The Common Gateway Interface (CGI) is an industry standard for method of communicating between a Web server and another program often used in such instances.




• Figure 2: Basic Components of a Generic Web Server

As with browsers, Web servers involve many different kinds of program components. Figure 2 illustrates the common components found in most servers. The key components present in all Web servers are a basic protocol machine for HTTP, a means to fetch Web pages or resource templates, and a mechanism to validate the composition of the response. The remaining components represent mechanisms to generate information dynamically. Besides the widely supported CGI standard, the specific choices depend on the Web server manufacturer.

A CGI application executes as a separate process and can be written in a variety of programming languages. As an independent process, the application is capable of accessing other hosts (e.g., a database server) and resources in performing its function, subject to its system security permissions. Once the application creates the information, the Web server conveys it in a response back to the browser. One drawback with this approach is that it consumes a significant amount of computational resources to spawn a new process for each request. The kernel receives an interrupt each time the application is called; then the application must be allocated memory, loaded into memory, passed the input parameters from the browser, and executed. If the application is called multiple times simultaneously from different sources, multiple copies of the application are resident in memory at the same time.

A number of other programming interfaces, such as the NetScape Server Application Programming Interface (NSAPI) and the Internet Server Application Programming Interface (ISAPI) from Microsoft, have been developed to communicate more efficiently between the software on the Web server and another program or library. An application built using one of these interfaces, however, operates quite a bit differently from a CGI application. For example, rather than executing as an external application in a separate process, an ISAPI application executes as an integral part of the Web server within the same address space as the server code. This is possible because the ISAPI application is a dynamic link library component rather than an external program. Moreover, because a dynamic link library can be loaded or unloaded at will, the ISAPI application can remain in memory, or be unloaded from memory to conserve system resources when it finishes and reloaded again if needed. An ISAPI application is also a shared multithreaded image, requiring only a single copy to support multiple simultaneous browser requests. These characteristics conserve system resources and improve response.

To improve Web server capabilities, product designers have developed a variety of techniques for dynamically generating content. Many of the techniques, such as applying scripting languages on the server side, are counterparts to those used within a browser. However, the techniques tend to be proprietary and used by software manufacturers to differentiate their product from others in the marketplace. The following items provide examples of a few of the more notable technologies.

### Server Side Includes (SSI)

SSI is a simple server side scripting language supported by most Web servers. SSI provides a set of dynamic features, such as including the current time or the last modification date of the HTML file, as an alternative to using a CGI program to perform the function. When the browser requests a document with a special file type, such as ".shtml", it triggers the server to treat the document as a template, reading and parsing the entire document before sending the results back to the client. SSI commands are embedded within HTML comments (e.g., <!--#include file="standard.html" -->). As the server reads the template file, it searches for HTML comments containing embedded SSI commands. When it finds one, the server replaces that part of the original HTML text with the output of the command. For example, the SSI command given above (i.e., #include file) replaces the entire SSI comment with the contents of another HTML file. This allows one to display a corporate logo or other static information prepared in another file in a uniform way across all corporate Web pages. A subset of the directives available in server-side includes allow the server to execute arbitrary system commands and CGI scripts, which may have unwanted side effects.

### Microsoft Active Server Pages (ASP)

ASP is a server-side scripting technology from Microsoft similar to SSI, which can be used to create dynamic and interactive Web applications. An ASP page is essentially an HTML page that contains server-side scripts that run when a browser requests an ".asp" file from the Web server. Both Jscript and VBScript are supported scripting languages, but other languages can be accommodated. The Web server processes the requested page and executes any script commands encountered, before sending the results to the user's browser. Scripting capabilities can be extended through ActiveX objects. A script invokes an Active X object causes the object to be created and supplied any needed input parameters. Note that ActiveX is an optional technology not required by Active Server Pages.

### Java Servlets

Servlets are based on Java technology – essentially, a kind of server-side applet. They involve a technique to determine whether the browser's request requires dynamically generated information from a servlet object on the server. If so, the Web server can obtain a specified servlet object corresponding to the request (e.g., uploaded from another server) and execute it to obtain the results corresponding to the request. Servlets support an object-oriented environment on the Web server, which is flexible and extendible. The Web server is populated with the servlet objects, which operate in a continual loop until invoked. Thus, there is no startup overhead associated with execution of the servlet objects. By observing a common applications program interface, the servlet objects can run in any server environment. Moreover, untrusted servlet objects can be executed in a secure area, with the dynamically generated information being passed from the secure area into the remaining server environment.

## Threats

The openness of the Internet makes it easily accessible to intruders. Over recent years, intruder activity has revealed a number of shortcomings in the original design of the Internet. While a number of security features were foreseen and built into the protocols, including the keystone Internet Protocol (IP), numerous others were not addressed. Intruders have used these omissions to attack hosts on the Internet. Moreover, the Internet amplifies risks associated with vulnerabilities in connected hosts by exposing those vulnerabilities to a broader range of threats.

An attack is a realization of some specific threat that presents a danger to the confidentiality, integrity, availability, or accountability of a computational resource. There are a number of generic security threats that apply to systems on the Internet: unauthorized release of information, modification of information, and denial of resource usage. The Web, as a substrate over the basic Internet technology, is subject to these same threats. In addition, the capabilities for supporting active content and mobile code provide new threat opportunities that

also fall within these general categories. Like any technology, active content can provide a useful capability, but can also become a source of vulnerability for an attacker to exploit. In general, there are three different classes of attacks against this framework:

- Browser Oriented: Attacks can be launched against Web browser components and technologies.
- Server Oriented: Attacks can be launched against Web server components and technologies.
- Network Oriented: Attacks can be launched against the network infrastructure used to communicate between browser and server.

Fred Cohen [Coh95] noted some fundamental security issues of the design of the Web, which bear repeating: "

- *Distributed untrusted computation:* As a basic premise, the Web provides a means for information provided by arbitrary servers at unknown locations operated by unknown organizations to be interpreted by any of a large number of different browsers at unknown locations operated by unknown organizations. The idea of interpreting unknown information from unknown sources seems inherently risky.
- *Remote execution of untrusted software:* Many Web extensions are designed to provide added function making the Web more than just a massive uncontrolled distributed database. These extensions, such as Postscript, Java, and Mime essentially allow for remote execution of untrusted software. For the browser, the risk is that the computer running the browser will be taken over, while for a server, the same risk extends to the server and any subsequent browsers that get information from that server once it is attacked.
- *Remote interpretation of unstructured and unverified content:* In essence, most browsers and servers assume that the incoming information follows the HTTP protocol, but there is inadequate enforcement of this by servers and browsers. The result is that any incoming information might not conform, might be interpreted using an undefined method (corresponding to a don't care condition in the interpreter), and might result in arbitrary undesirable side effects."

Active content can be considered to be a form of mobile code, whereby the code in the form of a script, macro, or other portable representation can move either indirectly (e.g., via an electronic mail attachment) or directly (e.g., via a Web page) from one platform to another where it eventually executes. Because the technology is designed to be seamless, a user is often unaware of what is happening. The problem with mobile code in general, and active content in particular, is that it can embed a Trojan horse or other form of malicious code into a system.

Security is perhaps the most significant obstacle to the widespread use and adaptation of mobile code technology. Unlike client-server or other distributed system architectures, the mobile code owner and the execution environment operator may be different. The situation is outside the scope of classical computer security, particularly regarding the threats stemming from attacks on the application by the execution environment

## Technology Related Risks

In order to protect computational resources from attack, appropriate countermeasures, such as hardware and software mechanisms, policies, procedures, and physical controls, must be in place. The absence of or weakness in a countermeasure is security vulnerability. Most computer technologies involve some degree of vulnerability due to flaws in the design or implementation of the hardware and software. While these vulnerabilities are often subtle and do not affect the functionality of a product, they can be discovered and exploited by an attacker. The impact of a vulnerability to an individual or organization is the subject of a risk

analysis, and can vary widely, depending on such factors as the value of the resource affected or the perceived harm to one's reputation.

As a means of understanding the risks involved in active content, some popular active content technologies and their associated vulnerabilities are described below. They are provided as recent examples and do not imply an endorsement or condemnation by NIST of the product or underlying technology. While most of them provide a useful capability when used in a Web environment, they have also been successfully exploited them in the past. The motivation for these technologies is to improve functionality and gain flexibility for the user. In a Web application, this often involves moving code processing away from the Web server onto the client's Web browser. Allowing remote systems to run arbitrary code on a local system, however, poses serious security risks. Traditional client-server systems do not involve such risks since they rely on static code on both the server and client sides.

### PostScript

One of the earliest examples of active content is PostScript document representation. PostScript is a page description language that is still in wide use today. PostScript commands are language statements in ASCII text that are translated into the printer's machine language by a PostScript interpreter built into the printer. PostScript is also capable of being interpreted by software on most computer platforms and drawing to computer screens or an attached drawing device. The interpreter uses scalable fonts, eliminating the need to store a variety of font sizes.

A Postscript file contains a document description, which is specified in the Postscript page description language. The language is a powerful interpreted language, comparable to many programming languages, and, therefore, Postscript documents inherently convey active content. For example, the language defines primitives for file manipulation, which can be used in a Postscript document to modify arbitrary files when the document is displayed or printed. Unfortunately the operations can be abused by intentionally embedding malicious file commands within an otherwise harmless image, so that in displaying the image the interpreter also causes damage.

An early exploit of Postscript technology involved the language's ability to set a password held by the interpreter. In some hardware implementations of the language interpreter, if the password was set, it remained in non-volatile memory and prevented subsequent documents from being printed unless they contained the same password. An attacker sending a password-setting document could disable the printer in this way, requiring hardware replacement to rectify the situation [Cle90, Spe90]. Some Postscript interpreters can be set to disable potentially harmful primitives, which can also inhibit useful functions. This dilemma is a recurring theme with active content.

### Java

Java is a full-featured programming language compiled into platform-independent byte code that is executed by an interpreter called the Java Virtual Machine (JVM). The resulting byte code can be executed where compiled or transferred to another Java-enabled platform (e.g., conveyed via an HTML Web page as an applet). Java is useful for adding functionality to Web sites and many services offered by various popular Web sites require the user to have a Java-enabled browser. When the Web browser sees references to Java code it loads the code and then processes it using the built-in JVM.

The developers of Java tried to address the problem of security and were successful to a large extent. The Java programming language and runtime environment [Gon98, Gos96] enforces security primarily through strong type safety. Java follows a so-called sandbox security model, used to isolate memory and method access, and maintain mutually exclusive execution domains. Java code such as a Web applet is confined to a sandbox, designed to prevent it from inspecting or changing files on a client file system and using network connections to circumvent file protections or people's expectations of privacy.

Hostile applets still pose security threats even while executing within the sandbox. A hostile applet can consume or exploit system resources in an inappropriate manner, or causes a user to perform an undesired or unwanted action. Examples of hostile applets exploits include denial-of-service, mail forging, invasion of privacy (e.g., exporting of identity, electronic mail address, and platform information) and installing backdoors to the system. The Java security model is fairly complex and can be difficult for a user to understand and manage, which can increase risk. Moreover, many implementation bugs have also been found, which allow one to bypass security mechanisms.

### JavaScript and VBScript

JavaScript is a general purpose, cross-platform scripting language whose code can be embedded within standard Web pages to create interactive documents. JavaScript, which is similar to Microsoft's Jscript, was developed by Netscape. Both are founded on the same standard, the ECMAScript Language Specification, ECMA-262 [ECMA99]. The scripting language is extremely powerful and able to perform anything a user can do within the context of the browser. Design and implementation bugs have been discovered in both commercial scripting products. JavaScript does not have methods for directly accessing a client file system or for directly opening connections to other computers besides the host that provided the content source.

Visual Basic Script (VBScript) is a programming language developed by Microsoft for creating scripts that can be embedded in Web pages for viewing with the Internet Explorer browser. Netscape Navigator, however, does not support VBScript. Like JavaScript, VBScript is an interpretative language able to processes client-side or server-side scripts. VBScript is a subset of the widely used Microsoft Visual Basic programming language and also works with Microsoft ActiveX Controls. The language is similar to JavaScript and poses similar risks.

In theory, confining a scripting language to boundaries of a Web browser should provide a relatively secure environment. In practice, this has not been the case. The main sources of problems have been twofold: the prevalence of implementation flaws and the close binding of the browser to related functionality such as an electronic mail facility or the underlying operating system. Past exploits include sending a user's URL history list to a remote site, and stealing the mail address of the user and forging electronic mail.

### ActiveX

ActiveX is a set of technologies from Microsoft that provide tools for linking desktop applications to the World Wide Web. ActiveX controls are reusable component program objects that can be attached to electronic mail or downloaded from a Web site. ActiveX controls also come preinstalled on Windows platforms. Unlike Java, which is a platform-independent programming language, ActiveX controls are distributed as executable binaries, and must be separately compiled for each target machine and operating system.

The ActiveX security model is considerably different from the Java sandbox model. The Java model restricts the permissions of applets to a set of safe actions. ActiveX, on the other hand, places no restrictions on what a control can do. Instead, ActiveX controls are digitally signed by their author under a technology scheme called Authenticode. The digital signatures are verified using identity certificates issued by a trusted certificate authority to an ActiveX software publisher. For an ActiveX publisher's certificate to be granted, the software publisher must pledge that no harmful code will be knowingly distributed under this scheme. The Authenticode process ensures that ActiveX controls cannot be distributed anonymously and that tampering with the controls can be detected. This certification process, however, does not ensure that a control will be well behaved. The ActiveX security model assigns the responsibility for the computer system's security to the user.

Before the browser downloads an unsigned ActiveX control, or a control whose corresponding publisher's certificate was issued by an unknown certifying authority, the browser presents a dialog box warning the user that this action may not be safe. The user can choose to abort the transfer, or may continue the transfer if they assume the source is trustworthy or they are willing to assume the risk. Users may not be aware of the security implications of this decision, which may have serious repercussions. Even when the user is well informed, attackers may trick the user into approving the transfer. In the past attackers have exploited implementation

flaws to cover the user dialogue window with another that displays an unobtrusive message such as "Do you want to continue?" while exposing the positive indication button needed to launch active content.

## Desktop Application Macros

Developers of popular spreadsheet, word processing, and other desktop applications created macros to allow users to automate and customize repetitive tasks. A macro is a series of menu selections, keystrokes, and commands that have been recorded and assigned a name or key combination. When the macro name is called or the macro key combination is pressed, the steps in the macro are executed from beginning to end. Macros are used to shorten long menu sequences as well as to create miniature programs within an application. Macro languages often include programming controls (IF, THEN GOTO, WHILE, etc.) that automate sequences like any programming language. A virus can be written into a macro that is stored in a spreadsheet or word processing document. When the document is opened for viewing or use, the macro is executed and the virus is activated. It can also attach itself to subsequent documents that are saved with the same macro. For these reasons, under normal circumstances desktop applications should not be configured to open automatically as a helper application for a browser.

The recent Melissa virus is an example of the potential risk involved. A Microsoft Word document containing a malicious Visual Basic for Applications (VBA) macro propagated itself through the Internet by sending the host document as an electronic mail attachment addressed to contacts found in the victim's address book. VBA is an integral part of MS Office applications, included as a means for developers to build custom solutions within that environment. VBA is a superset of VBScript and offers the same automation and customization capabilities, but within the context of a desktop application.

The newer generation of electronic mail applications, including the ones built into Web browsers, support HTML content and MIME attachments. Since active content provides a number of avenues for exploits, such enclosures should be opened only after due consideration of the inherent risks. The problem lies in the dual roles for which HTML is being used. On the one hand, HTML is surpassing ASCII as a common means for composing and exchanging documents. On the other hand, HTML is also being used as an environment to house such things as scripting languages, Java applets, and ActiveX components. By combining the flexibility to send and receive HTML content with its ability to embody scripts and other forms of programs that have full access to memory and files, the potential for abuse becomes apparent.

## Plug-ins

Plug-ins are programs that work in conjunction with software applications to enhance their capabilities. Plug-ins are often added to Web browsers to enable them to support new types of content (audio, video, etc.). Such plug-ins can be downloaded from either the browser vendor's site or a third party site. Browsers typically prompt the user to download a new plug-in when a document that requires functionality beyond the browser's current capabilities. Although plug-ins allow browsers to support new types of content, they are not active content in and of themselves, but simply an active-content-enabling technology. Windows Media player, RealPlayer, ThingViewer, QuickTime, ShockWave and Flash are all examples of plug-ins that allow browsers to support new content types ranging from audio, video, interactive animation, and other forms of "new media."

For example, the ShockWave plug-in from Macromedia provides the ability to render multimedia presentations created in a compatible format, as they are downloaded. By design, ShockWave content supports the Lingo interpretative language as an aid to presentation development. When creating a ShockWave presentation, the author can include custom code using Lingo. Early versions of Lingo allowed the author to make local system calls based on the platform executing the content, potentially allowing malicious code to be downloaded as part of the presentation.

From a security standpoint, plug-ins contain executable code and, therefore, precautions should be exercised in obtaining and installing them, as with any other software application. Downloading free software code and authorizing its installation by simply clicking an "Install now" or equivalent button is risky. Downloading plug-ins

from a reputable manufacturer can mitigate the risk, but even in this case it is difficult for the user to always be aware of the security implications. In the past, unwanted side effects such as changes to browser security settings and tracking of a user's content preferences, however well intentioned, have occurred. Plug-ins designed to animate cursors or hyperlinks have also been designed to better track user preferences and viewing habits across a particular Web site. Although these additional capabilities may improve the user's experience with a particular Web site, the privacy and security implications are often not readily disclosed. Even if the site has a valid identity certificate associated with the signed downloaded code, that only tells the user that the manufacturer of the code has been verified by a certificate authority, but not whether the code obtained from them will behave non-maliciously or correctly. Users of plug-ins should be cautioned to read the fine print before agreeing to download executables, and take adequate measures to backup the system in the event of problems.

### CGI

CGI applications can be written for a Web server in a number of programming languages. More often than not, a scripting language such as Perl is used for this purpose, because of its flexibility, compactness, and facility. If scripts are not prepared carefully, attackers can find and exercise flaws in the code to penetrate a Web server. Therefore, scripts must be written with security in mind and should not, for example, run arbitrary commands on a system or launch insecure (or non-patched) programs. An attacker can find such flaws through trial and error and does not necessarily need the script source code.

There are two general areas where CGI applications can create security vulnerabilities at the server:

- They may intentionally or unintentionally leak information about the host system that can aid an attacker, for example, by allowing access to information outside the areas designated for Web use.

- When processing user provided input, such as the contents of a form or a search command, they may be vulnerable to attacks whereby the user tricks the application into executing arbitrary commands supplied in the input stream.

Ideally, scripts should constrain users to a small set of well-defined functionality and validate the size and values of input parameters so that an attacker cannot overrun memory boundaries or piggy back arbitrary commands for execution. In the event that a script does contain flaws, it should be run only with minimal privileges (i.e., non-administrator) to avoid compromising the entire Web site. However, potential security holes can be exploited even when CGI applications run with low privilege settings. For example, a subverted script could have enough privileges to mail out the system password file, examine the network information maps, or launch a login to a high numbered port.

The two areas of vulnerability mentioned, potentially affect all Web servers. While these vulnerabilities have frequently occurred with CGI applications, other related interfaces and techniques for developing server applications have not been immune. CGI being an early and well-supported standard has simply gained more notoriety over the years, and the same areas of vulnerability exist when applying similar Web development technologies at the server.

## Countermeasures

A number of steps can be taken to mitigate the risks in using active content. Generally speaking, there are two avenues to follow: avoidance – staying completely clear of known and potential vulnerabilities, and harm reduction – applying measures to limit the degree of potential loss due to exposure. The following sections highlight some of the more useful countermeasures one can apply.

## Security Policy

A security policy is the set of rules, principles, and practices that determine how an organization implements its security. Information security in any organization is largely dependent on the quality of the security policy and the processes that an organization imposes on itself. No amount of technology can overcome a poorly planned, poorly implemented, or nonexistent security policy. If the policy is not stated clearly and consistently, and not made known and enforced throughout an organization, it creates a situation ripe for exploitation.

Therefore, having or establishing an organizational security policy is an important first step in applying countermeasures for active content. For example, an Internet security policy can address enabling Java, JavaScript, or ActiveX on individual user's Web browser in various ways:

- Functionality must be disallowed completely,
- Functionality is allowed, but only from internal organizational servers,
- Functionality is allowed, but only from trusted external servers, and
- Functionality is allowed from any server.

## Risk Analysis and Management

Security involves continually analyzing and managing risks. Any such analysis must identify vulnerabilities and threats, anticipate potential attacks, assess their likelihood of success, and estimate the potential damage from successful attacks. Risk management is the process of assessing risk, taking steps to reduce risk to an acceptable level, and maintaining that level of risk. One of the most significant security pieces missing from most organizations is an on-going practice of risk analysis and management.

Because security is relative to each organization, it must be tailored to an organization’s specific needs, budget, and culture. For example, an attack launched against one organization could succeed easily and compromise extremely important information, while on another organization would only result in minimal damage, perhaps because of an absence of sensitive data. Companies, much like people, have personalities with differing comfort levels on the amount of risk that is reasonable, which also influence this process. Once an assessment is made, countermeasures can be put in place against those attacks deemed significantly high by either minimizing the likelihood of occurrence or minimizing the consequences of the attack. Different countermeasures are employed to meet an organization’s specific needs.

## Computer Incident Response Handling

Regardless of how well executed an organization’s security program is, inevitably a breach in security does occur. Besides adopting reasonable precautions for securing systems and networks, one must also establish the ability to respond quickly and efficiently when a security incident occurs. A security incident is an adverse event or situation involving a networked information system that poses a threat to the integrity, confidentiality or availability of the information system. Examples of incidents include unauthorized use of an account, unauthorized elevation of system privileges, and execution of malicious code that corrupts data or other code. Incidents may result in a partial or complete loss of security controls, an attempted or actual compromise of data, or the waste, fraud, abuse, loss or damage of computational resources. Responding to computer security incidents in an effective manner requires significant amount of advance preparation. Incidence Response activities require technical knowledge, communication and coordination among personnel who respond to the incident, in order to return the system as quickly as possible to normal operations.

## Security Audit

An increasing popular approach for measuring the security posture of an organization is through a security audit. Audits ensure that policies and controls already implemented are operating correctly and effectively. Audits can include static analysis of policies, procedures, and countermeasures as well as active probing of the systems external and internal security mechanisms. The results of an audit identify the strengths and weaknesses of the security of the system and potentially provide a list of deficits for resolution rated by degree of severity. As the security posture of a system evolves over time, audits are most effective when done on a reoccurring basis

## Evaluated Software

Whenever possible, preference should be given to a product that has undergone a formal security evaluation versus one that has not. The focus of a security evaluation is primarily on the correctness and effectiveness of the design, under the well-founded principle that a sound design enables a secure implementation, but an unsound design is hopelessly doomed. Products that have undergone other less formal forms of third-party testing and evaluation are also preferable to those lacking such scrutiny.

Note that using successfully tested or evaluated software does not necessarily ensure a secure operational environment. Even when a product does successfully complete a formal security evaluation, it may contain vulnerabilities. In addition, the way in which a product is applied and composed with other system components also affects security.

## Application Settings

The desktop applications that handle documents containing active content typically have built-in controls that can be used to control or prevent access. For example, both Netscape and Microsoft Web browsers have an options menu that can be used to select appropriate security settings regarding active content within downloadable documents. Spreadsheet, word processor, and presentation graphic software applications have similar controls. Even today, many manufacturers deliver products with insecure default settings. It behooves the user of such applications to become familiar with the security options available and use them in accordance with organizational policy.

## Automated Filters

If malicious content has been identified and understood, it can be detected and eliminated or rejected completely from entering. For example, many firewalls are capable of filtering electronic mail attachments for well-known file types, such as .exe executable files, and deleting them at the point of entry. More sophisticated filters can perform checks for viruses within executables and hidden macros within document files. Anti-virus software has also become increasing capable of detecting electronic documents having active content with a malicious code signature. Such software can be applied at the Web browser or at a proxy running at a cache manager.

## Version Control

Users can gain better security by applying security patches as soon as they become available. This is a well-known and effective remedy, but for a variety of reasons also a well-ignored one. Users can also take advantage of security enhancements to their applications by upgrading to newer versions. Microsoft Windows 98 users can use the Windows update feature to find bug fixes and product updates and download them automatically from the WWW. Using this feature, however, requires the user to use code that will scan the computer for installation information in order to properly install any upgrades. The Microsoft Web site states that none of this information is sent to Microsoft or over the Internet. Updating software products automatically over the Web is becoming increasingly popular, as the benefits to the user are considerable. As this practice becomes more commonplace, users must be better aware of the implicit decisions they make when allowing

vendors to run software on their machine. For example, updating an audio player on a computer may allow the vendor to track the user's musical preferences.

### Readers

Occasionally, manufacturers of desktop applications provide free software readers, which are capable of interpreting their proprietary file formats, for recipients of the documents who do not own the application. The Adobe Acrobat Reader, for example, allows users to view and print Portable Document Format (PDF) files, but does not allow users to edit them. Since the software readers are only intended to produce a readable rendition of the document and are not full fledge applications, they bypass many potentially harmful features and exploits based on implementation vulnerabilities contained in a specific application. Besides manufacturer-provided readers, general-purpose software readers are commercially available, which are capable of rendering dozens of file formats. A related measure is the selection of documents with less capable forms of active content, when multiple choices are offered. Some Web sites offer an electronic document in a variety of formats such as native word processor format, Postscript, or PDF. While PDF represents text and graphics using the imaging model of the PostScript language, PDF is not a programming language and contains no language constructs, making it the safest alternative.

### Isolation

Isolation can be applied at various levels. The simplest is complete isolation at the system level. A production computer system that is unable to receive documents containing active content cannot be affected by malicious hidden code. Although it is not always possible to isolate a system physically, logical isolation (e.g., via router settings or firewall controls) may be applied, at least to some degree. For example, risky functions, such as Web browsing, may be confined to a second system designated exclusively for that purpose. Often older or spare systems are available and could be put to good use this way.

Isolation of tightly bounded proprietary program components is another alternative. Seamless interoperation of products such as electronic mail, Web browsers, and office applications is a goal of product manufacturers. In order to provide better functionality or performance, manufacturers often allow products within their product line to take advantage of little known or undocumented interfaces, which has from time-to-time lead to unwanted or insecure side effects. By integrating products from different manufacturers, one can effectively isolate program components from using all but the documented and standard interfaces.

## Summary

Active content documents offer several benefits to both the users of these documents and their authors. Java applets, JavaScript, VBScript and ActiveX provide additional functionality to Web pages, plug-ins enable browsers to support new types of content, Postscript offloads the processing and interpretation of the presentation of documents to the printer, and macros automate repetitive word processing and spreadsheet tasks. HTML, JavaScript and Java are relatively platform Independent and can run on both Internet Explorer (IE) and Netscape Navigator. VBScript and Javascript can also be used to pass information between HTML, Java, and ActiveX Components.

The benefits of each of these active content technologies must be carefully weighed against the risks they pose. Security is not black or white, but shades of gray. When employing active content technology, security measures should be put in place to reduce risk to an acceptable level and to recover if an incident occurs.

Informed security officers, administrators, and other IT professionals are responsible for developing security policies based on their organization's specific security needs and level of acceptable risk. Unfortunately, there is rarely a "one size fits all" guideline that fits the unique needs of every organization and each organization must decide what constitutes an acceptable level of risk. Establishing an organizational security policy is an important

step in developing and applying appropriate security measures. The IT and security staff have a responsibility for keeping abreast of the associated risks with emerging technologies by subscribing to security mailing lists and visiting vendor Web sites for information and updates for products used within their organization. As active content moves beyond desktop personal computers to mobile handsets, television sets, and a wide variety of other consumer electronic goods, users will be faced with competing and difficult tradeoffs between privacy and security, with increased functionality and ease-of-use.

## References


[Anu98] V. Anupam, A. Mayer, Secure Web Scripting, IEEE Internet Computing, vol. 2, no. 6, November/December 1998, pp. 46-55.

[Cle90] Robert E. Van Cleef, “New Roque Imperils Printers,” The Risks Digest, Volume 10, Issue 32, September 1990, http://catless.ncl.ac.uk/Risks/10.32.html

[Coh95] Fred Cohen. "Internet Holes: 50 Ways to Attack Your Web Systems," Management Analytics, 1995. http://all.net/journal/netsec/9512.html

[Gon98] L. Gong, Java Security Architecture (JDK 1.2), Draft Document, revision 0.8, Sun Microsystems, March 1998, http://service.felk.cvut.cz/doc/java/share/jdk1.2beta3/docs/guide/security/spec/security-spec.doc.html

[Gos96] J. Gosling, H. McGilton, The Java Language Environment: A White Paper, Sun Microsystems, May 1996, http://SunSITE.sut.ac.jp/java/whitepaper/

[Hug99] Paul Hughes,”Building A Web Browser,” THT Productions Inc., December 1999, http://www.vbweb.co.uk/controls/web_browser.htm

[Spe90] Henry Spencer, “Re: New Roque Imperils Printers,” The Risks Digest, Volume 10, Issue 35, September 1990, http://catless.ncl.ac.uk/Risks/10.35.html

[Ven99] Venners, Bill, “A Walk Through Cyberspace,” JavaWorld, December 1999, http://www.javaworld.com/javaworld/jw-12-1999/jw-12-jiniology.html.

[Wer99] Werring, Laurentius, “ The Hidden Threat Within,” 11th Annual Canadian Information Technology Security Symposium, May 1999, pp. 201-214.

[ECMA99] ECMAScript Language Specification, 3rd edition, Standard ECMA-262, December 1999, ftp://ftp.ecma.ch/ecma-st/Ecma-262.pdf

[HTTP] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1", IETF Network Working Group, RFC 2616, June 1999. http://www.ietf.org/rfc/rfc2616.txt?number=2616

[MIME] N.Borenstein and N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies, IETF Network Working Group, RFC 1521, September 1993. http://www.ietf.org/rfc/rfc1521.txt?number=1521

[CSS1] “Cascading Style Sheets – level 1 (CSS1),” W3C Recommendation, January 1999, http://www.w3.org/TR/1999/REC-CSS1-19990111

[CSS2] “Cascading Style Sheets – level 2, (CSS2),” W3C Recommendation, May1998, http://www.w3.org/TR/1998/REC-CSS2-19980512

[XML1] “Extensible Markup Language (XML) 1.0,” Second Edition, W3C Recommendation, October 2000, http://www.w3.org/TR/2000/REC-xml-20001006

[HTML4] "HTML 4.01 Specification," W3C Recommendation, December 1999, http://www.w3.org/TR/html4/html40.txt